RNOTES

The Journal for Users of
the RPNZL Programming System
and Applications

Issue 1 February-March 1984

WELCOME

I'd like to welcome you to the first issue of RNOTES, a semi-monthly publication for users (and non-users) of the RPNZL Programming System for the ZX81 and TS1000 computers. In this first issue are several articles that explain just what RPNZL is and just what it can do.

Although RPNZL has been available for less than a year, it has become an important addition to the supply of software for the

continued on page 2

MISSILE GRAPHICS

THE CORE of any computer graphics animation is the problem of simulating the motion of objects, or "missiles". The listings below will show how this is done in RPNZL.

LISTING 1

```
▒MISSILE1                01.08.84
START 7D00 FORGET
(   5)
(  10)  SEG SUB
(  20)  1710 A "
(  30)  0 1700 100 STEPDO
(  40)    A £ POSN @ SEND
(  50)    1700 INDX -
(  60)    10 OR DUPE
(  70)    POSN A "
(  80)    CH A SEND
( 100)  LOOP
( 110)  END
( 120)
( 130)
```

continued on page 3

DESIGN PHILOSOPHY

THE RPNZL Programming System has been designed to provide an alternate environment for the ZX81 and TS1000 computers. This design was carried out with four primary goals in mind:

1) to increase the speed of program execution;

2) to reduce program and data memory requirements;

3) to give the programmer greater access to the hardware; and

4) to be compact enough to allow implementation of a full system within a 16K RAM environment. **R**

RPNZL QUARTET

RPNZL (Reverse Polish Notation Zx81 Language) is a fast machine code program that interprets RPNZL codes, or tokens, mucha as the ROM in the TS1000/ZX81 interprets BASIC tokens. There are many differences in how this is done, but mainly RPNZL is faster, more compact and more versatile.

Included in the RPNZL System Package (see page 5) are four essential programs written in RPNZL. The first is the Monitor and is laoded along with RPNZL itself within a BASIC program. Its function is to load, save and verify data to tape 12 times faster than BASIC; plus, to let

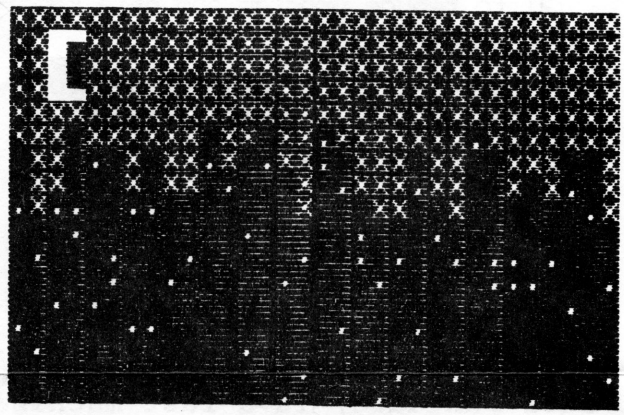
continued on page 4

(continued from page 1)

little machine. With RPNZL operating in 16K of RAM, the computer suddenly grows in stature, allowing sophisticated programming in the small environment. In memories beyond 16K, BASIC's non-structured, non-extensible nature rapidly becomes a liability, while RPNZL becomes even easier to use. If you've ever written even a short BASIC program, you know how difficult it can be to find a bug; in 32K+ the problem becomes insurmountable.

Of all the programming aids, utilities, toolkits, as well as language implementations, for the TS1000, RPNZL alone has completely revised the operating system to include high-speed, software-only cassette file system, full I/O redirection (I/O to/from files, printer,screen), built-in graphics commands, and more!

RNOTES is devoted to bringing you more information about all these features, so stay with us. **R**



```
SKYLINE
START 7D00 FORGET
DEF SUB1 PWD HERE SEG SUB
SWITCH X £ OR >PRINT END
DEF SUB2 PWD HERE SEG SUB
R £ - RAN R £ + SUB1 PROC END
SEG SUB
0 1F DO INDX X "
  5 RAN 0A + DUPE R "
  15 DO
     $ "■". INDX SUB1 PROC
  LOOP
  $ "■". 16 SUB2 PROC
  $ "■". 11 SUB2 PROC
  5 R £ 1 - DO
     $ "■". INDX SUB1 PROC
  LOOP
LOOP
$ T02" ▄"CT02" ■"CT02" ▀".
0600 >PRINT
END
STOP
```

R

MECHANICS OF RPNZL

A RPNZL program consists of a series of bytes called tokens, each representing a program step. Each token is fetched in turn and processed, the system variable (SYSVAR) Code Pointer (CPT) being used to point to the next token, much as the Program Counter is used in machine code.

The tokens are divisible into groups:

ØØ-7F	Operators
8Ø-Bl	Modules
B2-CB	Local variables
CC-FF	Global variables

As each token is fetched its highest bit, bit 7, is tested; if it is zero (token < 8Ø) it falls into the first group. The tokens value doubled is used as an offset into the token table. At the referenced location is the address of the machine code routine that actually performs the function of the token. For example, token Ø9 (DUPE) doubled is 12, plus 4124 (the base of the token table) is 4136; at 4136 is the address of the actual DUPE routine, 438Ø.

Tokens 6C through 7E, plus ØE, 37 and 3F, are not presently used; their slots in the token table simply point to the Error 9 (Code Error) routine.

Tokens with bit 7 a one have that bit reset to zero, then the token is doubled to form as offset used as follows:

The module tokens 8Ø through Bl serve as one-byte subroutine calls. The offset formed from the token points into the module table (itself pointed to by the SYSVAR MODTBL). At the referenced location is another offset from the base of the module table pointing to the subroutine called by the module token.

The variable tokens B2 through CB and CC through FF are one-byte variable address references. The offset is added to the contents of the SYSVARs GLOBVTBL or LOCVTBL; the resulting variable address is placed on the stack. The local variable tokens are designated :A to :Z; the global variables are A to Z and AA through ZZ.

The use of single-byte codes to refer to the most used operators, subroutines and variables is the key to RPNZL's compactness. **R**

(continued from page 1)

In Listing 1 the problem is solved as economically as possible. Load and Run EDCOM(P), execute Kill File (UK), then enter Listing 1 in a text file; Compile it, then link it with the deBug command, and then Run it. The "missile", a capital A, should "take off" from the middle of the bottom of the screen and stop when it reaches the first screen line.

It moves pretty fast, huh? If you did not see it, check the text file for errors, then Run it again, and watch carefully. Try the same program in BASIC sometime.

Here's how it works, line by line:

The text file's title line starts with a grey square (CHR$ 8, hex Ø8) to form the text file name's length byte. The last 24 characters of the line is ignored by the Compiler, leaving room for the date (or whatever).

The START address is set to 7DØØ and the dictionary is cleared by FORGET.

Line 1Ø declares a code segment, line 2Ø initializes global variable A with the starting position of the missile (row 17 hex, column 1Ø hex; 23,16 decimal).

Line 3Ø sets up a "DO-LOOP" (like BASIC's "FOR-NEXT") that counts from Ø to 17ØØ by 1ØØs (all numbers hex). This has the effect of stepping the loop's index through all the screen row numbers in RPNZL's coordinate notation. This notation consists of the row expressed as the high byte and the column expressed as the low byte of a single integer.

Line 4Ø takes care of erasing the old missile.

LISTING 2

```
▒MISSILE2                01.08.84
START 7D00 FORGET
( 10)
( 20) DEF ORIGIN PWD 1700
( 30) DEF COLUMN PWD 0010
( 40) DEF OLD.POSITION VAR A
( 50)
( 60) SEG SUB
( 70) ORIGIN COLUMN OR
( 80) OLD.POSITION "
( 90) 0 1700 100 STEPDO
(100)   OLD.POSITION £ POSN
(110)   0 SEND
(120)   ORIGIN INDX -
(130)   COLUMN OR DUPE
(140)   POSN OLD.POSITION "
(150)   CH A SEND
(160)   2 DELAY
(170) LOOP
(180) END
```

Line 5Ø inverts the loop index so that the row numbers will run from 17ØØ through Ø by 1ØØs. This is done because RPNZL will not allow a loop limit to be exceeded by the loop index; negative step values are thus not permitted.

Line 6Ø merges the column with the row number, and duplicates it on the stack. Line 7Ø sets the print position to the new one, and stores the duplicate in A, so that line 4Ø can find the missile to erase it on the next iteration. Line 8Ø then prints the new missile.

Line 1ØØ loops back to line 4Ø to repeat the process for each line.

Go back to the text file and insert a line 9Ø as follows:

```
( 9Ø)       2 DELAY
```

ReCompile, deBug, and run. The delay provided by line 9Ø, lasting 2/60ths of a second on each row, makes the action more visible.

Enter listing 2 as shown; Compile, deBug and Run. The effect is the same, but the program seems more readable. This is because of the 2 constants, ORIGIN and COLUMN, and the global varaible A, renamed OLD.POSITION.

ORIGIN is the missile's line of origin, and COLUMN is the screen column in which the missile will travel.

Lines 7Ø and 8Ø of Listing 2 accomplish the same thing as line 2Ø of Listing 1.

The remaining lines of Listing 2 duplicate the DO-LOOP of Listing 1; the DEFined words simply function as stand-ins for the desired values.

Listing 3 shows how this problem would be coded modularly. If you examine the final (program) SEGment, you will see that is even more readable.

All the elements of the original program are still there, except they are now represented symbolically.

INITIALIZE-- This sets up the global variable OLD.POSITION just as line 2Ø of Listing 1 does; plus, the shape of the missile (and the unmissile) are stored as strings, and variables are initialized with their addresses, so that $PRINT can be used in MAKE.MISSILE and ERASE.MISL.

continued on page 4

(Missile Graphics, cont. from page 3)

LISTING 3

```
MISSILE3                01.08.84
START 7D00 FORGET

( 10)  DEF OLD.POSITION VAR A
( 20)  DEF ORIGIN PWD 1700
( 30)  DEF COLUMN PWD 0010
( 40)  DEF TOP BYT 07
( 50)  DEF WAIT PWD 0001
( 60)
( 70)  DEF MISSILE VAR B
( 80)  DEF MSL PWD HERE
( 90)  SEG $CONS "▗▟▙▖".
(100)
(110)  DEF MAKE.MISSILE PWD HERE
(120)  SEG SUB
(130)  POSN MISSILE $PRINT END
(160)
(170)  DEF UNMISSILE VAR C
(180)  DEF SPC PWD HERE
(190)  SEG $CONS "    ".
(200)
(210)  DEF ERASE.MISL PWD HERE
(220)  SEG SUB
(230)  OLD.POSITION @ POSN
(240)  UNMISSILE $PRINT
(250)  END
(260)
(270)  DEF SET.POSITION PWD HERE
(280)  SEG SUB
(290)  COLUMN OR DUPE
(300)  OLD.POSITION ! END
(320)
(330)  DEF SLOW.DOWN PWD HERE
(340)  SEG SUB
(350)  WAIT DELAY END
(370)
(380)  DEF INITIALIZE PWD HERE
(385)  SEG SUB
(390)  ORIGIN SET.POSITION PROC
(395)  DROP
(400)  MSL MISSILE !
(410)  SPC UNMISSILE !
(420)  END
(430)
(435)  DEF NEW.POSITION PWD HERE
(440)  SEG SUB
(445)  ORIGIN INDX - END
(450)
(455)  SEG SUB
(460)  INITIALIZE PROC
(480)  TOP ORIGIN 100 STEPDO
(500)    ERASE.MISL PROC
(510)    NEW.POSITION PROC
(520)    SET.POSITION PROC
(530)    MAKE.MISSILE PROC
(540)    SLOW.DOWN PROC
(550)  LOOP
(560)  END
(570)
```

QUANTITY PRICES

The RPNZL System Package, as described on page 5, is regularly $29.95. For quantities of 10 to 24, there is a 25% discount; for 25 or more the discount is 50%. We pay shipping on orders of 5 or more.

THE GOLDEN STAIR
141A Dore Street
San Francisco, CA 94103
(415) 552-1415

The DO-LOOP is set up using TOP (referring to the top line) and ORIGIN (referring to the bottom line). Within the loop ERASE.MISL is used; this gets the OLD.POSITION and prints the "UNMISSILE" there.

NEW.POSITION-- This procedure inverts the line number value of the loop index.

MAKE.MISSILE-- Here, the "MISSILE" is printed at the position stored in the variable NEW.POSITION.

SLOW.DOWN-- This has been included to show how a constant (as in this case), a variable, or a procedure returning a value can be used to slow down the loop in order to make the missile more visible.

As can be seen, the modular style makes the most readable code. Using the program SEGment as given in Listing 3, completely new subroutines to draw, erase and recalculate the missile can be designed and implemented. **R**

(continued from page 1)

you examine, change, and print data in your computer's memory. The Monitor also supervises the running of other RPNZL programs.

The next two programs come together in one file on tape; they are the Editor and Compiler (EDCOM). The Editor allows you to type into the computer just like a typewriter, except a "text file" is used instead of paper. The Editor also lets you load, save, verify and print these text files.

The Compiler takes a text file prepared with the Editor and translates it into RPNZL codes, or tokens, and stores the code in "link files".

Later, a number of link files are all played back to the fourth program, called the Linker, which (appropriately) links the code together into a program.

Even if you're not a programmer, with the RPNZL System you can run any RPNZL programs others have written, and still take advantage of the added speed and power. **R**

—CONTRIBUTED ARTICLES WELCOME—

RPNZL IN LARGER MEMORY ENVIRONMENTS

ALTHOUGH designed to be usable in a 16K system, RPNZL shows its real capabilities in environments beyond 16K. Due to hardware limitations, it is not possible to run machine code in the upper half of memory (8ØØØ-FFFF), nor may the display file straddle the 8ØØØ address mark. However, RPNZL, as long as no machine code is included, works without hindrance.

Figure 1 shows how RPNZL's work area is apportioned by the standard Editor/Compiler (EDCOM). Figure 2 diagrams an additional 16K.

When EDCOM is installed, a table containing the mapping data is copied into the external variables USR1 through USR12 at 4ØD8, where the Editor, Compiler and Linker can access it. This table is located at 745D and is 18 hex bytes long. After installation, the area is overwritten by the text file. Figure 3 shows the values and names of the twelve variables for the standard map of Figure 1 and for the expanded version of Figure 2.

Load EDCOM(P) at 65ØØ and then use the W(rite Mem) Monitor command to enter the new mapping data into the table at 745D. RAMTOP should be at 8ØØØ. The area 72EØ-7CFF is available for utility routines (string search, etc.) or whatever.

With the text file expanded to 13K from 2K, and the link file to 3K from ½K, program development is greatly simplified. **R**

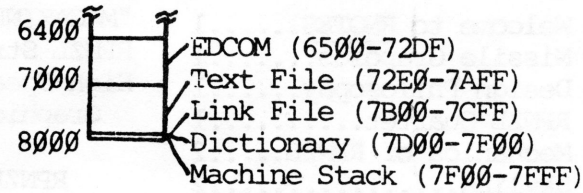


FIGURE 1

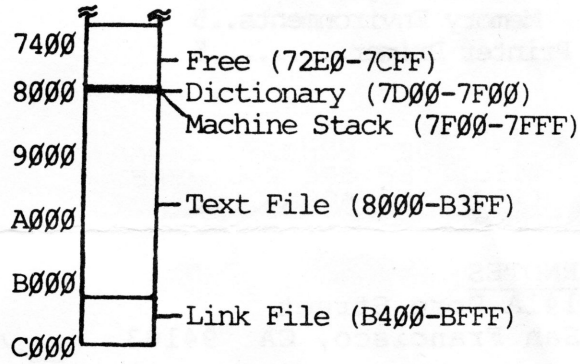


FIGURE 2

		Standard	Expanded
745D:	MAXLIST	ØØ26	Ø17A
5F:			
61:	TEXT FILE	72EØ	8ØØØ
63:	MAIN DFILE		
65:			
67:	NAME	72E1	8ØØ1
69:	LINES	ØØ3E	Ø192
6B:	LINK FILE	7BØØ	B4ØØ
6D:	DICT		
6F:	LPSTK		
71:	LINK LEN	Ø1FF	ØBFF
73:	TEXT LEN	Ø81Ø	34ØØ

FIGURE 3

PRINTER PRIMER

--PRINTED TAPE DIRECTORIES--
--PRINTED MEMORY DUMP--

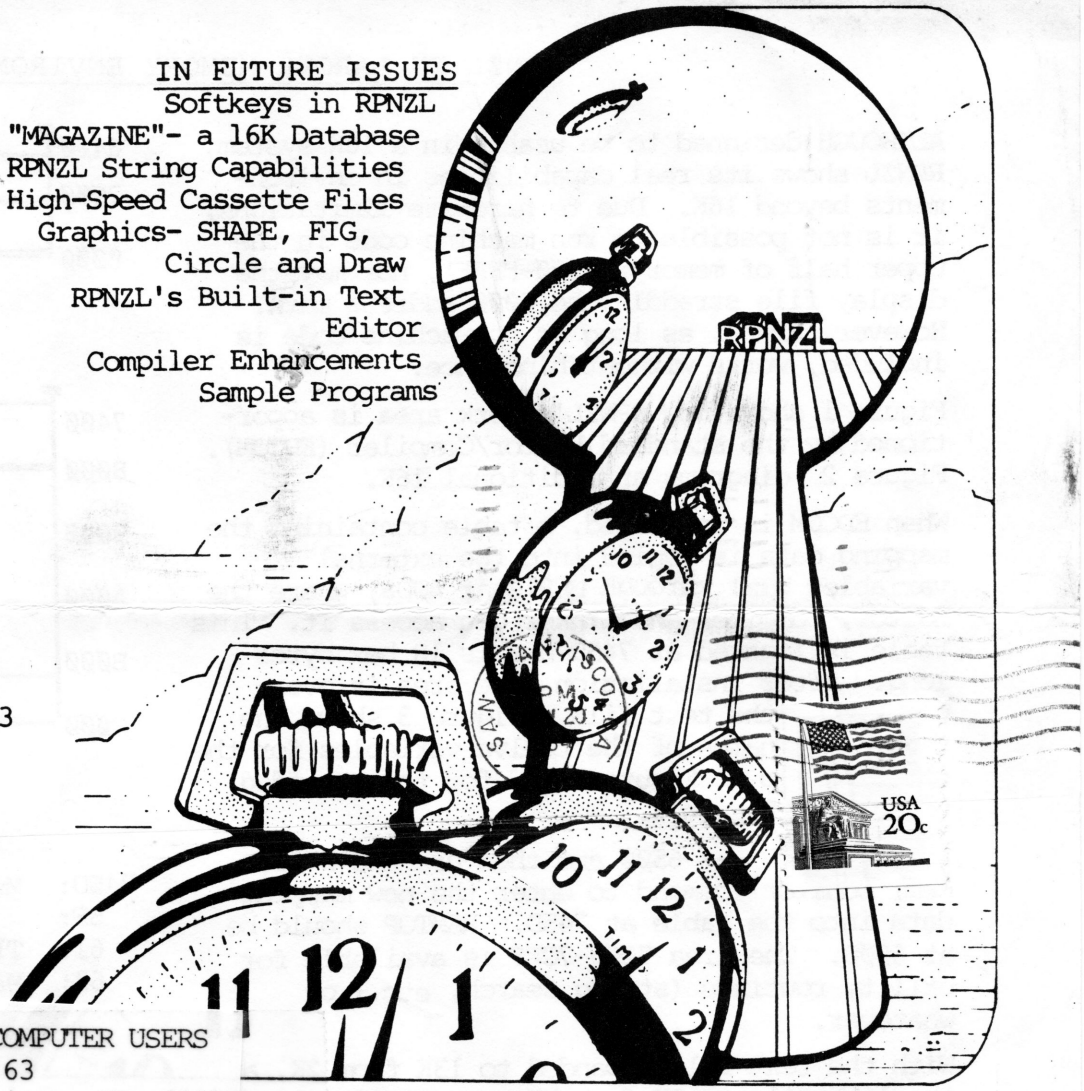
From the Monitor, use the W(rite Mem) command to place the value Ø3 at address 4Ø7C (OUTSTAT). This enables the printer. Use the F(ile Directory) or D(ump) commands in the normal fashion—each line is echoed to the printer! W(rite Mem) ØØ into address 4Ø7C to disable the printer again.

Within a program, the printer may be turned on by Ø1 OPEN and off again by Ø1 CLOSE. **R**

IN THIS ISSUE

IN FUTURE ISSUES

Softkeys in RPNZL
"MAGAZINE"- a 16K Database
RPNZL String Capabilities
High-Speed Cassette Files
Graphics- SHAPE, FIG,
Circle and Draw
RPNZL's Built-in Text
Editor
Compiler Enhancements
Sample Programs



RNOTES
141A Dore Street
San Francisco, CA 94103

TIMEX COMPUTER USERS
PO Box 63
Des Plaines, IL 60017

RNOTES Is a Semi-Monthly
Journal for Users of the
RPNZL Programming System for the
TS1000 and ZX-81 Computers